SISTEMI OPERATIVI

ESERCIZIO N. 1 del 18 FEBBRAIO 2000

Un ristorante dispone di **PMAX posti** e si avvale della presenza di **C camerieri**. I **clienti** arrivano a gruppi composti da 1 a PMAX/2 persone, e un gruppo entra nel locale solo se c'è un numero sufficiente di posti. I gruppi di clienti possono essere di due tipi: **abituali** e **occasionali**; i gruppi di clienti abituali hanno la priorità su quelli occasionali nell'ingresso al ristorante. Una volta seduti, i gruppi di clienti ordinano il pasto al primo cameriere libero, mangiano e poi escono dal ristorante. I camerieri attendono di essere chiamati dai gruppi di clienti, prendono l'ordinazione e poi portano i piatti al tavolo dei clienti. (Si suppone che i piatti siano sempre a disposizione in numero infinito).
Si implementi una soluzione usando il costrutto monitor per modellare il ristorante e i processi per modellare i gruppi di clienti e i camerieri, e si descriva la sincronizzazione tra i vari processi. Si discuta se la soluzione proposta può presentare starvation e in caso positivo per quali processi, e si propongano modifiche e/o aggiunte per evitare starvation.

Nota: per semplicità, non si è considerato il numero del tavolo a cui sono seduti i clienti.

```
program gestione_ristorante

const PMAX = ...; { numero di posti }
const C = ...; { numero di camerieri }

type   tipo = (A, O);  { Abituale e Occasionale }
       persone = 1..(PMAX/2)

type cliente = process(t: tipo, p: persone)
begin
      risto.entra(t, p);
      risto.ordina;
      <mangia>
      risto.esci(p);
end

type cameriere = process
begin
      repeat { in questo caso è necessario un ciclo infinito }
            risto.prendi_ordinazione;
            <porta l'ordinazione in cucina>
            <prendi i piatti>
            risto.porta_piatti;
      until false
end

type ristorante = monitor

{ variabili del monitor }
var  posti_occupati: integer;
      { numero di posti occupati }
      incoda: array [tipo] of integer;
      { numero di processi in coda per tipo }
      cam_occupati: 0..C;
```

```
        { numero di camerieri occupati }
        coda_fuori: array [tipo] of condition;
        { code su cui sospendere i clienti che aspettano di
        entrare }
        attesa_clienti: condition;
        { coda su cui sospendere i clienti che aspettano un
        cameriere }
        attesa_camerieri: condition;
        { coda su cui sospendere i camerieri che aspettano
        clienti }
        attesa_piatti: condition;
        { coda su cui sospendere i clienti che aspettano i piatti }

procedure entry entra (t: tipo; p: persone)
begin
        if t = A { clienti abituali }
        then
                while posti_occupati + p > PMAX
                { controllo solo la capacità }
                do
                begin
                        incoda[A]++; coda_fuori[A].wait; incoda[A]--;
                end;
        else { t = O  clienti occasionali }
                while posti_occupati + p > PMAX or
                { devo anche dare la precedenza agli abituali }
                        coda_fuori[A].queue
                do
                begin
                        incoda[O]++; coda_fuori[O].wait; incoda[O]--;
                end;

        posti_occupati := posti_occupati + p; { sono entrato }
end
```

```
procedure entry ordina
begin
	if cam_occupati = C
	{ se tutti i camerieri sono occupati }
	then
		attesa_clienti.wait;
	else { vedi considerazioni }
		attesa_camerieri.signal;
	{ dopo aver ordinato aspetto i piatti }
	aspetta_piatti.wait;
end

procedure entry esci (p: persone)
var s, i: integer;
begin
	posti_occupati := posti_occupati – p;
	{ risveglio prima i clienti abituali }
	if coda_fuori[A].queue
	then
	begin
		s := incoda[A];
		for i := 1 to s do
			coda_fuori[A].signal;
	end
	{ risveglio anche i clienti occasionali }
	{ penseranno loro a controllare la coda degli abituali }
	if coda_fuori[O].queue
	begin
		s := incoda[O];
		for i := 1 to s do
			coda_fuori[O].signal;
	end
end
```

```
procedure entry prendi_ordinazione
begin
      if not attesa_clienti.queue
      { se non ci sono clienti che devono ordinare }
      then
            attesa_camerieri.wait;
      else { vedi considerazioni }
            attesa_clienti.signal;
      cam_occupati ++ ;
end

procedure entry porta_piatti
begin
      cam_occupati -- ;
      aspetta_piatti.signal;
end


begin { inizializzazione delle variabili }
      posti_occupati := 0;
      incoda[A] := 0;
      incoda[O] := 0;
      cam_occupati := 0;
end

var  risto: ristorante; { il nostro monitor }
      cla1, cla2, ... : cliente (A, k);
      clo1, clo2, ... : cliente (O, j);
      c1, c2, ..., cC : cameriere;

begin end.
```

Considerazioni

Il risveglio dei camerieri/clienti deve essere fatto in alternativa alla sospensione, poiché, non sapendo chi dei due risveglia l'altro, fare due signal comprometterebbe la correttezza della soluzione.

Nella soluzione è stato usato un contatore (cam_occupati) per vedere se ci sono o meno camerieri liberi. Un'alternativa all'uso del contatore sarebbe stato il controllo della coda attesa_camerieri: se infatti non ci sono camerieri sospesi, significa che sono tutti occupati, mentre un cameriere sospeso è libero.

Poiché il processo cliente non fa niente tra le invocazioni delle procedure **entra** e **ordina**, sarebbe stato anche possibile mettere il codice di entrambe le procedure in un'unica procedura **entra_ordina**.

Starvation

La soluzione proposta presenta starvation, infatti è possibile che i clienti abituali non lascino entrare i clienti occasionali.

Un modo per evitare starvation è quello di utilizzare dei contatori, in modo che dopo che sono passati P clienti abituali, si facciano entrare i clienti occasionali.